PCT WORLD INTELLECTUAL PROPERTY ORGANIZATION
International Bureau

INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification 6 : G06F 17/60	A2	(11) International Publication Number: WO 99/06929
		(43) International Publication Date: 11 February 1999 (11.02.99)

(21) International Application Number: PCT/US98/12809

(22) International Filing Date: 3 August 1998 (03.08.98)

(30) Priority Data:

Not furnished	3 August 1997 (03.08.97)	US
60/054,648	4 August 1997 (04.08.97)	US

(71) Applicant: AT & T CORP. [US/US]; 32 Avenue of the Americas, New York, NY 10013-2412 (US).

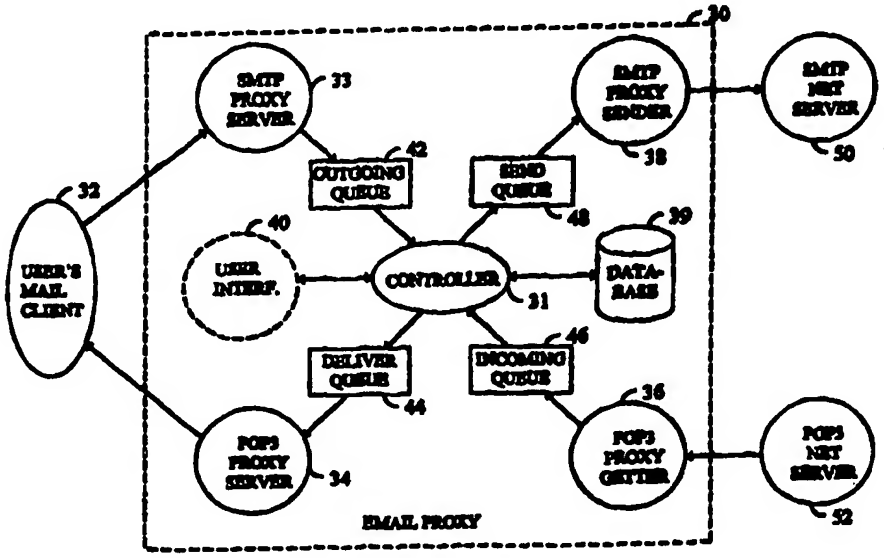
(72) Inventor: HALL, Robert, J.; 79 Beech Avenue, Berkley Heights, NJ 07922 (US).

(74) Agent: DWORETSKY, Samuel, H.; AT & T Corp., P.O. Box 4110, Middletown, NJ 07748 (US).

(81) Designated States: CA, European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE).

Published
Without international search report and to be republished upon receipt of that report.

(54) Title: AN EXTENSIBLE PROXY FRAMEWORK FOR E-MAIL AGENTS



(57) Abstract

A system for processing electronic messages comprises a communications port for exchanging electronic messages with an electronic mail network. The system also includes an electronic mail client and a proxy application resident in a memory. The proxy application has a configurable controller specifying a manner of processing electronic messages received from the electronic mail client and the communications port. A processor within the system, coupled to the communications port and the memory, executes the proxy application to process the electronic messages received from the electronic mail client and the communications port in accordance with a configuration of the controller.

FOR THE PURPOSES OF INFORMATION ONLY

Codes used to identify States party to the PCT on the front pages of pamphlets publishing international applications under the PCT.

AL	Albania	**ES**	Spain	**LS**	Lesotho	**SI**	Slovenia
AM	Armenia	**FI**	Finland	**LT**	Lithuania	**SK**	Slovakia
AT	Austria	**FR**	France	**LU**	Luxembourg	**SN**	Senegal
AU	Australia	**GA**	Gabon	**LV**	Latvia	**SZ**	Swaziland
AZ	Azerbaijan	**GB**	United Kingdom	**MC**	Monaco	**TD**	Chad
BA	Bosnia and Herzegovina	**GE**	Georgia	**MD**	Republic of Moldova	**TG**	Togo
BB	Barbados	**GH**	Ghana	**MG**	Madagascar	**TJ**	Tajikistan
BE	Belgium	**GN**	Guinea	**MK**	The former Yugoslav Republic of Macedonia	**TM**	Turkmenistan
BF	Burkina Faso	**GR**	Greece			**TR**	Turkey
BG	Bulgaria	**HU**	Hungary	**ML**	Mali	**TT**	Trinidad and Tobago
BJ	Benin	**IE**	Ireland	**MN**	Mongolia	**UA**	Ukraine
BR	Brazil	**IL**	Israel	**MR**	Mauritania	**UG**	Uganda
BY	Belarus	**IS**	Iceland	**MW**	Malawi	**US**	United States of America
CA	Canada	**IT**	Italy	**MX**	Mexico	**UZ**	Uzbekistan
CF	Central African Republic	**JP**	Japan	**NE**	Niger	**VN**	Viet Nam
CG	Congo	**KE**	Kenya	**NL**	Netherlands	**YU**	Yugoslavia
CH	Switzerland	**KG**	Kyrgyzstan	**NO**	Norway	**ZW**	Zimbabwe
CI	Côte d'Ivoire	**KP**	Democratic People's Republic of Korea	**NZ**	New Zealand		
CM	Cameroon			**PL**	Poland		
CN	China	**KR**	Republic of Korea	**PT**	Portugal		
CU	Cuba	**KZ**	Kazakstan	**RO**	Romania		
CZ	Czech Republic	**LC**	Saint Lucia	**RU**	Russian Federation		
DE	Germany	**LI**	Liechtenstein	**SD**	Sudan		
DK	Denmark	**LK**	Sri Lanka	**SE**	Sweden		
EE	Estonia	**LR**	Liberia	**SG**	Singapore		

AN EXTENSIBLE PROXY FRAMEWORK FOR E-MAIL AGENTS

FIELD OF THE INVENTION

The present invention relates to a proxy framework coupled between an electronic mail client and a network for processing electronic messages. Specifically, the proxy framework performs functions on behalf of the mail client while preserving a standardized interface to a network having electronic messaging capability.

BACKGROUND OF THE INVENTION

There has recently been a massive proliferation of interconnected or networked computers. One of the biggest advantages of this proliferation is the ability of users at computers coupled to the network to exchange electronic mail or electronic messages. Existing electronic mail allows users to exchange information between computers connected by a modem, connected across a local area network, or connected across the Internet. In the latter configuration, groups of computers having electronic mail capability are typically coupled to the internet through an internet service provider (ISP) or gateway. Electronic messages can convey not only text information, but also images, sounds, motion picture with sound and binary files with executable programs.

Advanced electronic mail features can significantly enhance the user's experience of electronic mail by reliably and tirelessly performing mundane tasks associated with features like classification, filtering, encryption, authentication, and auto response. Conventionally some advanced features have been implemented as extensions to (or replacements for) particular e-mail client programs. However, implementation of advanced features within the client electronic mail software itself is cumbersome and inconvenient. This is because of (1) the number and diversity of

client software packages, (2) the fact that the code of client software packages is typically beyond the control of the ISPs wishing to add features uniformly to all clients within their service groups, and (3) the difficulty of persuading users to switch client software.

Some advanced features have also been implemented at servers located in the network path between the many e-mail client programs and the remote source of the e-mail on the network. However, these "server-side" implementations suffer from either limiting the preferences of the individuals who maintain electronic mail capability coupled to the server, or from the impracticality of administering personalized solutions on a large scale.

It would be desirable to implement agent-like capabilities having advanced functionality in the form of an e-mail proxy that operates logically between the off the shelf mail client, such as NETSCAPE e-mail and EUDORA e-mail, and network based mail servers. It would also be desirable to add advanced agent functionality arbitrarily, using a flexible, extensible proxy framework, without forcing users to change mail client software, server software, or any other aspect of their existing environment.

SUMMARY OF THE INVENTION

According to the present invention, a system for processing electronic messages includes an electronic mail client and a proxy application resident in a memory and a communications port for exchanging electronic messages with an electronic mail network. The proxy application has a configurable controller specifying a manner of routing electronic messages between the electronic mail client and the communications port. A processor within the system, coupled to the communications port and the memory, executes the proxy application to route electronic messages between the electronic mail client and the communications port in accordance with a configuration of the controller.

The controller of the proxy may be configured to perform any convenient function, such as filtering or terminating electronic messages meeting a

predetermined criterion or generating new messages in response to receiving an electronic message (autoresponse). In addition, the proxy may include interfaces that conform to standard protocols such as the simple mail transfer protocol (SMTP) for interfacing with the electronic mail client and a network node within the electronic mail network. Such standard interfaces facilitate integrating the proxy into the electronic message datapath between the mail client and the electronic mail network.

BRIEF DESCRIPTION OF THE DRAWINGS

The above described features and advantages of the present invention will be more fully appreciated with reference to the appended figures and detailed description.

FIG. 1 depicts a client computer having electronic mail capability coupled to a plurality of other client computers over a computer network.

FIG. 2 depicts a proxy framework according to the present invention.

FIG. 3 depicts a method of receiving electronic messages according to the present invention.

FIG. 4 depicts a method of transmitting an electronic message according to the present invention.

FIG. 5 depicts a method of processing user commands according to the invention.

DETAILED DESCRIPTION OF THE INVENTION

FIG. 1 depicts a plurality of client computers 10, each possessing electronic mail capability, interconnected over a computer network 26 allowing the exchange of data between client computers. Computer network 26 includes a plurality of network nodes 24 which may be interconnected electronically, optically, or wirelessly. Each network node 24 may have a plurality of client computers 10 coupled to it. Network 26 may be a local area network, a wide area network, or a network of computers operating under the Internet protocol. The network may

include one or more gateway nodes 28. The gateway nodes 28 translate between electronic mail messages in one protocol to electronic mail messages in another protocol used by the e-mail programs at the client computers 10 coupled to the gateway node 28.

FIG. 1 also shows an exploded view of a client computer 10. The client computer 10 includes a processor 12 coupled over one or more buses 14 to a memory 16, a keyboard 18, a mouse 19, a display 20, and a communications port 22. The bus 14 enables the exchange of data between the processor 12 and each interconnected component. The memory 16 includes both random access memory ("RAM") and fixed memory, for example, disk drives and optical drives. The memory 16 includes stored program instructions for a user's mail client 32 as well as stored program instructions for an electronic mail proxy 30 according to the present invention. The contents of the memory 16 are more fully described with reference to FIGS. 2-5. The processor 12 executes program instructions stored in the memory 16 and monitors and responds to instructions entered at the keyboard 18. In addition, the processor 12 updates the display 20 according to program instructions. The communications port 22 allows the processor 12 to communicate with other client computers 10 coupled to the network 26. The communications port 22 is implemented as a modem, a local area network card, or similar device for signaling over a communications link. The communications port 22 may be coupled over a local area network such as an Ethernet or token ring network to a network node 24. Alternatively, the communications port 22 may be connected to an Internet service provider node over a telecommunications link. It is through the communications port 22 that the processor 12 is able to send and receive electronic mail messages with other client computers on the network 26.

FIG. 2 shows the e-mail proxy architecture 30 and its interaction with a user's mail client 32. The e-mail proxy consists of six daemon processes including SMTP proxy server 33, POP3 proxy server 34, POP3 proxy getter 36, SMTP proxy sender 38, user interface 40, and controller 31. The proxy 30 also includes four persistent mail queues called outgoing queue 42, deliver queue 44, incoming queue

46, and send queue 48. The controller 31 acts as the brain of the proxy 30, directing the desired proxy functionalities such as filtering, auto response, encryption, authentication, and classification.

The controller 31 gets input from two of the four mail queues, incoming queue 46 and outgoing queue 42. Typically, the controller 31 will accept a message from one of these two queues, process it, and then place messages in one or both of its output queues, send queue 48 and deliver queue 44. Optionally, the controller 31 may send events to a user interface 40. The behavior of the controller 31 may be customized to include advanced features to meet the needs of a particular user.

The user customizes the controller at compile time (using a programming model "API" as will be described later in this disclosure) by specifying custom feature logic that comprises instructions for what the proxy is to do in response to the various incoming messages and other events it must handle.

For example, to implement a filtering proxy, the user would specify a set of rules that classify incoming messages based upon syntactic criteria defined by the user, such as "is the sender address that of a known correspondent? Does the message contain the word 'advertisement,'" or a synonym or close misspelling thereof? The user then surrounds these rules with procedural program steps that actually compute the syntactic criteria (via utility functions provided in the proxy framework, or perhaps by new utility functions coded by hand for this application) and then evaluate the rule predicates. The programmer includes steps that dictate if the predicates evaluate to true for a rule, then the action described by the rule is taken, such as delivering a message to the user or sending one out to the network.

As a concrete example, suppose the user wishes to filter out all messages containing the word "advertisement" in the subject line, so that they are never presented by the client to the user for reading. The user would specify program steps to the controller 31 that first applied a string search utility function (supplied by the framework in this case) to the incoming message, searching for the string "advertisement". Next, if the search result was negative, i.e. no such string was found, the feature logic would tell the proxy to place the message in the deliver

queue using the DELIVER act. Otherwise, if the search result was positive, it would dictate taking no further action; in particular, it would not place the message in any queue, so it would be discarded or terminated instead of being delivered to the client software.

As another example, to implement an encrypting/decrypting proxy, the user would specify to the controller 31 program steps that first extract the sender of an incoming message from the message header; next, look up the decryption key stored in the proxy database for that sender; next supply it, along with the encrypted message body to a decryption algorithm utility function (either provided by the framework or coded by hand), packaging the resulting decrypted body along with the original header information into a new message; and finally specifying the action of delivering this new message to the user. For outgoing messages, a similar sequence of steps could encrypt the outgoing message and send it to the intended recipient.

As an example, suppose a user has one correspondent with whom he shares an encryption/decryption key pair, and this correspondent sends him an encrypted email message. Once this message is presented to the custom controller, the feature logic for this controller would proceed as follows. The first step would retrieve the sender address from the header of the message using a supplied utility function. Next, the logic would do a LOOKUP of the DECRYPTION-KEY for that sender. If one is found, the next step would apply the decryption utility function to the message body using the looked up key. Finally, a new message would be constructed having the same header information as the original but having the decrypted body in place of the original body, and this new message would be placed in the deliver queue (using the DELIVER act) for presentation to the client. If no decryption key were found for an incoming message's sender, then the incoming message itself would simply be placed in the deliver queue (via the DELIVER act) without alteration. A similar scenario would apply to outgoing messages, using the ENCRYPTION-KEY state relation and the recipient field of the message, except

that multiple new (encrypted) messages may be created if the outgoing message has multiple recipients.

To implement a digital signature verification feature, the controller 31 may be programmed, for example, to examine an incoming message (using utility functions provided by the framework or coded by hand) to see if it contains a digital signature block (a section of text having a particular predefined format). If it does have such a block, then the sender's address is used as a key to look up in the state database a signature verification key, such as the decryption half of an asymmetric public/private key pair as is known in the art. If such a key is known, then a digital signature verification utility routine is applied to the message to determine whether the signature is valid. If it is, then, for example, a header line can be added that indicates "sender signature verified". Otherwise, a line could be added that indicates "sender signature not verified". Alternatively, a message with an unverified signature could be simply discarded by the proxy. If no key is known for a sender, a header line could be added to indicate "no sender key known".

To implement an autoresponse feature, the controller 31 may be programmed, for example, to examine incoming messages (using utility functions provided by the framework or coded by hand) to see if they have a particular structured command format. If an incoming message does not have this format, then it is simply delivered without change to the user's mailbox. If it does have this format, then a prescribed action can be taken, such as formatting a new message having the requested document as its body and addressing it to the sender address of the original message. The user-supplied feature logic would then specify that the new message is to be sent to that address.

As an example, suppose a user wishes to have his proxy automatically send out an electronic copy of any one of his published papers when any correspondent sends a request for one. And suppose a correspondent sends a message whose first body line is "SEND email-proxy.ps". The feature logic would direct the controller 31 to first examine this first line of the incoming message and detect adherence to the required format. It would then also check its state database to determine

whether "email-proxy.ps" is the name of a paper it has access to. If both checks succeed, it would create a new message, addressed to the sender of the original message, having a body consisting of the electronic format of the email-proxy.ps paper. It would finally place this new message in the send queue (using the SEND act) for subsequent transmission to the network. If either of the syntactic checks mentioned above fails, then the original message is simply placed unchanged in the deliver queue using the DELIVER act.

A proxy 30 may be configured to perform a plurality of the features described herein (for example, autoresponse, digital signature verification, and encryption) on the same message if desired.

Referring to Fig. 2, The SMTP ("simple mail transfer protocol") server is coupled to the user's electronic mail client 32 and to the outgoing queue 42. The SMTP server accepts outgoing messages from the user's e-mail client 32 using an SMTP protocol, which is well known, and places them in the outgoing queue 42. The POP3 server 34 is coupled to the delivery queue 44 and to the user's e-mail client 32. The POP3 server serves messages from the deliver queue 44, which is filled by the controller 31, to the user's mail client 32 according to the well known POP3 ("post office protocol") protocol. Both SMTP and POP3 are widely used on the Internet today. However, as new mail or electronic messaging protocols come into use, the SMTP server 33 and the POP3 server 34 may be replaced with the new protocols without affecting implementation of the other components of the proxy 30. The SMTP proxy sender 38 and the POP3 proxy getter 36 may similarly be replaced.

The SMTP proxy sender 38 is coupled to the send queue 48 and an SMTP network server 50 located on a network node remotely situated from the proxy 30. The SMTP proxy sender 38 monitors the send queue 48, which is filled by the controller 31, and uses the SMTP protocol to transmit these messages out to the network SMTP server 50 designated by the user's configuration. The user's configuration is stored in the database 39. POP3 proxy getter 36 is coupled to a POP3 netserver 52 located on a remote network node and to the incoming queue

46. The POP3 proxy getter 36 is a client for the network POP3 netserver 52 designated by the user in the user configuration stored in database 39. The POP3 proxy getter 36 gets messages from the POP3 netserver 52 and places them in the incoming queue 46 when prompted by a signal from the user's mail client 32 which is forwarded by the POP3 proxy server 34.

A user interface 40 may be added optionally. The user interface 40 is coupled to the controller 31 and allows the user direct control over the proxy 30. The user interface 40 may send and receive signals to and from the controller 31.

Control or processing within the proxy 30 is event driven. When an electronic message arrives in a queue, the queues' consumer process is awakened and the message stored. Storing a message into the queue may cause other processes to be awakened. For example, when a message arrives from the user's client via the SMTP proxy server 33, the SMTP proxy server 33 places the message into the outgoing queue 42, which wakes up the controller 31. In response, the controller 31 may place one or more messages into the send queue 48. Storing of the message in the send queue 48 may wake up a process within the SMTP proxy sender 38, causing the SMTP proxy sender 38 to send one or more messages from the send queue 48 to the SMTP netserver 50.

An extra signaling path is used to handle the on-demand nature of the POP3 protocol. When the mail client probes the proxy POP3 server 34, the latter sends a signal to the POP3 proxy getter 36 and waits a short (configurable) period before checking the deliver queue 44. This wait allows the POP3 proxy getter 36 a chance to retrieve any new messages from the POP3 network server 52. The wait also allows the controller 31 a chance to process any new messages retrieved, possibly placing one or more messages into the deliver queue 44. After the wait, the POP3 server 34 forwards any messages in the deliver queue 44 to the mail client 32. Alternatively, the proxy 30 can be configured to have the POP3 proxy getter 36 periodically check the network POP3 server 52 (even absent probes from the client), allowing proxy processing to proceed even when the user is not active at the mail client 32.

All of the message queues are persistent, which means that the data within each queue is stored with fixed memory backup such as on a disk drive or an EEPROM, so that in the event of a system crash the contents of the queues will not be lost. Messages are not deleted from one queue until the consumer process for the queue has finished processing and either has stored result messages into persistent queues (in the case of the controller) or confirmed successful completion of a protocol (message delivery by SMTP sender 38 or POP3 server 34). There is a small risk of duplicate message processing if a crash occurs after the result messages are written and before the input messages are deleted; however, this is arguably a better tradeoff than the opposite approach which allows the possibility of losing messages.

The proxy 30 may be implemented using any software language including c, c++, PASCAL and COBOL. In a preferred embodiment, the proxy 30 is implemented as a stand-alone Java application in a reusable, object-oriented fashion within the client computer 10 . Typically, the proxy is started up once (for example when the machine 10 is booted). There are at least two startup options. The initialize-start option performs all initializations, including starting over from an empty state database. The warm-start option performs only enough initialization to resume processing where processing left off the last time the machine went down. Typically, the user will perform an initialize-start once and then warm-starts each boot thereafter. On initialization, the proxy application reads a configuration within the database 39 file that defines several parameters, such as the hostnames and ports of the SMTP and the POP3 network server 50 and 52 respectively and the location on the local disk where the persistent queues are to be stored.

FIG. 3 illustrates a method of handling incoming electronic mail messages by the proxy 30. It also illustrates the interaction between the proxy 30 and the user's mail client 32. In step 100, a user at the computer 10 issues a command to the mail client 32 to get any new mail present on the network 26. The user may indicate this with a keystroke on the keyboard 18, or with a click of a mouse coupled to the computer 10, or in any other convenient manner of indicating commands to the

computer 10. In response to the "get mail" command, in step 102 the user's mail client 32 sends a request to the POP3 proxy server 34 within the proxy 30 to get any new mail which may have arrived at the proxy 30 from the network. In step 103, the POP3 proxy server 34 forwards a signal for getting mail to the POP3 proxy getter 36. In response to the signal, the POP3 proxy getter 36 issues a request to the POP3 netserver 52 specified by the user's configuration in the database 39. The request is for new mail. In addition, the proxy POP3 server 34 suspends in step 104. In step 105, the POP3 proxy getter 36 receives from the POP3 netserver 52 any new messages designated for a computer 10. In step 106, the POP3 getter 36 sends received messages to the incoming queue 46 and to the controller 31.

In step 107, the controller 31 processes the incoming event and places any messages received from the POP3 proxy getter 36 into the deliver queue 44 when the message is for the user's e-mail client and the controller 31 updates its state via one or a collection of set commands to the database 39. Optionally, if the controller configuration so specifies, then no updates may be made via set. The controller 31 may forego placing the received message into the deliver queue 44 depending on whether the controller 31 has been configured with advanced features. For example, if the controller is actively filtering incoming messages and the incoming message meets the filtering criteria, the message is terminated without being forwarded to the email client. In addition, if the controller 31 is implementing autoresponse, the controller 31 may, in response to receiving an incoming message, place a response message in the send queue 48 pursuant to the feature logic and configuration information in database 39. In general, arbitrary numbers of response messages may be placed in the send and deliver queues as prescribed by the user specified feature logic. In step 108, the proxy 30 optionally updates its user interface window with messages descriptive of the controller's actions to inform the user of such actions. In step 109, the POP3 proxy server 34 comes out of its suspended state and checks the deliver queue 44 for any new messages which have arrived. If new messages have arrived, in step 109, the POP3 proxy server 34 forwards the messages to the user's mail client 32. In step 110, the mail client

presents new messages to a user. In step 111, the controller's new state is saved to the database 39. In step 112, the SMTP sender 38 sends a message from the send queue 48 to a SMTP network server 50 which forwards the message to the new address determined by the feature logic, incoming message content, and configuration information in the send queue. Step 112 is optional and will not occur unless the feature logic dictates one or more response messages.

FIG. 4 illustrates a method for sending a message from the mail client through the proxy 30 to a remote mail client on the network. In step 200, a user composes a message in the mail client 32. The user then initiates a "send operation" when the message has been completed. In step 202, the mail client 32 sends the message to the SMTP proxy server 33 within the proxy 30. In step 204, the SMTP proxy server 33 receives the message from the user's mail client 32 and stores it into the outgoing queue 42. Also in step 204, the SMTP server 33 indicates to the controller 31 that the message has been stored in the outgoing queue 42. In step 206, the controller 31 processes the outgoing event and places the message presently in the outgoing queue 42 into the send queue 48. If the controller 31 dictates, no messages will be placed in the send queue 48. In addition, the controller 31 updates its state via one or more "set" commands and stores its state in the database 39. This is indicated in step 210. In step 208, the proxy optionally updates a user interface window within the display 20 to indicate the status of the controller. In step 212, the SMTP proxy sender 38 transmits the message from the send queue 48 to the SMTP netserver 50 over the network. The message is in turn further transmitted over the network 26 to the remote address specified in the e-mail header. Additionally, the controller 31 may be customized with a feature logic causing one or more messages to be placed in the deliver queue 44 when the outgoing message meets a predefined criteria. The predefined criteria may be based on content contained within the header or the body of the outgoing e-mail message, as well as the current state of the controller 31. For example, the controller state may change based on the time of day or other conditions as specified by the customizer.

FIG. 5 illustrates a method for processing user commands. In step 300, a user issues a command to the proxy 30 using the user interface window 40. In step 302, the proxy user interface 40 receives and sends the command to the controller 31. In step 304, the controller 31 processes the incoming command and places messages into the send 48 and/or deliver queue 44 when the command so requires. Additionally, the controller 31 may update its state within the database 39 when required. In step 306, the proxy 30 updates the proxy user interface 40 when the command has been processed. In step 308, the controller's new state is saved. In step 310, the SMTP proxy sender 38 sends messages from the send queue 48 to the SMTP netserver over the network 26 when so required by the user issued command.

In a preferred embodiment of the invention, the proxy 30 is customized, i.e. the desired feature logic is specified at compile time, by specializing three classes. The proxy class is the highest level such class. It has two overridable factory methods, make-controller and make-interface, which return instances of the other two user-specialized classes, controller and notifiable-window. (It is not required for customization to override make-interface, if no interface window is needed.) Another optional override is the init-from-config method which allows custom initializations to be performed, based (in part) on information read from the configuration file within database 39.

The preferred implementation provides an abstract controller class. In order to provide the desired feature logic at compile time, the customizer overrides the following five methods to implement the desired agent functionalities (all except the last have void return types):

- handle_INIT () is called once at initialize-start only
- handle_INCOMING (message, user). This method is called once for each message placed in the incoming queue 46 by the POP3 getter 36 on behalf of a user.

- handle_OUTGOING (message, user). This method is called once for each message placed in the outgoing queue 42 by the SMTP proxy server 33 on behalf of a user.
- handle_COMMAND (command, user, arglist). This method may be called by the user interface 40 in response to a user gesture. It may also be called at initialization time if desired.
- required_db_properties (). This method is called at initialization time to set up the proxy's state database 39.

The proxy callers of the "handle_x" methods are preferably synchronized, so that a programmer needn't worry about concurrency.

Each of these specialized methods may call effectors (acts), state database operations, and utility routines, which are provided by the framework code. Two effector methods are provided: MAIL (message) deposits a message into the send queue 48, and DELIVER (message) deposits a message into the deliver queue 44.

A simple (persistent) database model is provided to record the state of the proxy 30 which is saved to disk and restored if the proxy is warm-started. An overloaded SET method allows setting values of database relations, which may be 1, 2, or 3 placed relations in the current implementation. The required-db-properties method must return a list of db-property objects defining these relations. (This effectively defines the database schema.) An overloaded LOOKUP method is provided to read the database relations. For example, the 2-place relation "CORRESPONDENT-ADDRESS" could implement a simple address book relation useful for expanding aliases. Then SET ("CORRESPONDENT-ADDRESS", "Bob", "hall@research.att.com") would set its values and LOOKUP ("CORRESPONDENT-ADDRESS", "Bob") would retrieve them. The proxy infrastructure makes these operations persistent, using database techniques that are well known in the art.

A library of utility routines is provided for analyzing and synthesizing messages, such as extracting the message sender, canonicalizing e-mail addresses, or constructing a new message from sender, recipients, and text. The abstract data

type "message," implemented using database techniques that are well known in the art, preferably hides the detail involved in making messages persistent, so that the user has a simple programming model with which to work.

The abstract notifiable-window class, instances of which are typically user-interface windows, must implement the methods notify-set, notify-event, and notify-act which are called when (a) a state-db relation is modified (SET), (b) an event, such as INCOMING, OUTGOING, COMMAND, or INIT, occurs and is handled, and (c) an act effector is invoked, such as MAIL or DELIVER. Thus, the user-interface 40 can react when any of these occur. Notifiable-window implementations may invoke INCOMING, OUTGOING, COMMAND, or INIT events during processing. Typically, this processing will accept user gestures, issue commands to the controller 31, and update a status display.

In one embodiment of the invention, the proxy 30 is configured to implement a personal channel agent (PCA) which manages a user's e-mail channels. The PCA performs address translation such as rewriting a header within the e-mail message and bookkeeping, for example, storing extended correspondent return addresses in the database 39 and correlating each return address to a particular piece of outgoing mail. The PCA also autonomously engages in a channel switching protocol with another PCA instance running somewhere else in the network. This protocol allows PCA's, by sending structured e-mail messages to each other, to agree on which channel is to be used by the users at the PCA for addressing mail between them.

In order to prohibit malicious users from impersonating a user, a configuration parameter may be included that restricts the SMTP server to accepting network connections only from the same host it is running on. This is necessitated by the insecurity of the SMTP protocol, which does no authentication. With this restriction, the proxy must run on the user's machine. However, this restriction can be relaxed if the proxy is to run in the network, possibly by dispatching on the IP address of the incoming connection and connecting to a proxy devoted to users known to have that IP address.

To customize the proxy 20 to implement the PCA, I extended the proxy and controller classes, as described above, including implementing an interface window allowing the user to directly command the proxy to do various things. I had to write only 1700 lines of Java code custom for the PCA as compared to the 4600 lines in the preferred embodiment of the proxy framework. Thus, about 73% is reused framework code. Of course, reuse fraction is sensitive to message processing complexity of the task.

In addition to the PCA implementation, the proxy 30 has other more general applications. For example, in another embodiment, the proxy 30 can be customized to make a general mail forwarder/gateway with arbitrary filtering, authentication, and autoresponse functionalities, by using only the SMTP half of the proxy (deactivating the POP3 portions by setting a switch). For example, by using only the SMTP proxy sender 38 and (an enhanced version of) the POP3 proxy server "half" of the proxy 30, one can implement a multi-user host with filtering, autoresponse, etc, where multiple users can pick up their mail. One can also make this user-programmable with a user proxy communicating to the host via e-mail messages. The integrated SMTP interface allows the easy collaboration of multiple proxy-like agents distributed around the network. Thus, the proxy framework can be used as a platform for building distributed software agent applications.

Although specific embodiments of the present invention have been disclosed, one of ordinary skill in the art will appreciate that changes can be made to those embodiments without departing from the spirit and scope of the invention.

CLAIMS

What is claimed is:

1. A system for processing electronic messages at a computer coupled to an electronic mail network, comprising:

a communications port for exchanging electronic messages with an electronic mail network;

an electronic mail client resident in a memory;

a proxy application resident in the memory, the proxy application having a configurable controller specifying a manner of processing the electronic messages received from one of the electronic mail client and the communications port; and

a processor coupled to the communications port and the memory, the processor executing the proxy application to process the electronic messages received from one of the electronic mail client and the communications port in accordance with a configuration of the controller.

2. The system according to claim 1, further comprising:

a first interface within the proxy application, the first interface coupling the electronic mail client to the proxy application pursuant to at least one network protocol.

3. The system according to claim 2, wherein the first network interface includes a SMTP proxy server and a POP3 proxy server.

4. The system according to claim 2, further comprising:

a second interface within the proxy application, the second interface coupling the electronic mail network to the proxy application pursuant to the at least one network protocol.

5. The system according to claim 4, wherein the second network interface includes a SMTP proxy sender and a POP3 proxy getter.

6. The system according to claim 5, wherein the first network interface includes a SMTP proxy server and a POP3 proxy server.

7. A system according to claim 1, wherein the controller is configured to terminate a first one of the electronic messages arriving at the proxy when the first electronic message meets a predetermined criteria.

8. A system according to claim 7, wherein the predetermined criteria is that the first electronic message includes a predetermined string.

9. A system according to claim 8, wherein the first electronic message includes a subject field and the message is only terminated when the subject field of the message includes the predetermined string.

10. A system according to claim 1, wherein the controller is configured to generate a new electronic message for transmission to the electronic mail network through the communications port in response to receiving one of the electronic messages having a predetermined criteria.

11. A system according to claim 10, wherein the new electronic message is a text file stored in the memory.

12. A system according to claim 10, wherein the new electronic message is a binary file stored in the memory.

13. A system according to claim 1, wherein the proxy application further comprises:

queues for storing the electronic messages in the memory.

14. The system according to claim 13, wherein the queues store the electronic messages prior to the configurable controller routing the electronic messages.

15. The system according to claim 14, wherein the queues are persistent.

16. The system according to claim 14, wherein the proxy application further comprises:

interfaces for exchanging the electronic messages with the electronic mail client and the electronic mail network using a predetermined protocol.

17. The system according to claim 16, wherein the predetermined protocol is the SMTP protocol.

18. The system according to claim 16, wherein the predetermined protocol is the POP3 protocol.

19. The system according to claim 1, wherein the controller is configured to perform at least one of encrypting and decrypting the electronic message.

20. The system according to claim 1, wherein the controller is configured to perform digital signature verification on the electronic messages.

21. A computer program product for implementing a proxy in a computer, comprising:

a computer useable medium having computer program logic stored therein, wherein the computer program logic comprises:

interfaces for enabling the proxy to receive electronic mail messages from an electronic mail client and an electronic mail network;

at least one queue coupled to the interfaces for enabling the proxy to store the electronic messages received from one of the interfaces; and

a configurable controller for enabling the proxy to process the electronic messages received from one of the interfaces in accordance with a configuration of the controller.

22. The computer program product according to claim 21, wherein the configuration of the controller causes the proxy to terminate a first one of the electronic messages received when it includes a predetermined string.

23. The computer program product according to claim 22, wherein the first electronic message received includes a subject field and the proxy only terminates the first electronic message when the subject field includes the predetermined string.

24. The computer program product according to claim 21, wherein the configuration of the controller causes the proxy to generate a new electronic message for transmission to the electronic mail network.

25. The computer program product according to claim 21, wherein the interfaces implement a predetermined protocol.

26. The computer program product according to claim 25, wherein the predetermined protocol is the SMTP protocol.

27. The computer program product according to claim 25, wherein the interfaces include a SMTP proxy server and a POP3 proxy server.

28. The computer program product according to claim 27, wherein the interfaces further include a SMTP proxy sender and a POP3 proxy getter.

29. The computer program product according to claim 21, further comprising:

a database, coupled to the controller, specifying the configuration of the controller.

30. A method for processing electronic mail messages in a proxy application resident in a computer memory, the proxy application coupling an electronic mail client resident in the memory to an electronic mail network, comprising the steps of:

receiving at the proxy application an electronic mail message from one of the electronic mail client and the electronic mail network;

storing by the proxy application the electronic mail message; and

processing the electronic mail message at the proxy in accordance with a configurable configuration.

31. The method according to claim 30, further comprising the step of:

terminating the electronic message when it includes a predetermined string.

32. The method according to claim 31, wherein the electronic message includes a subject field and the terminating step only terminates the electronic message when the subject field includes the predetermined string.

33. The method according to claim 30, further comprising the step of:

generating a new electronic message for transmission to the electronic mail network in response to receiving the electronic mail message.

34. The method according to claim 30, further comprising the step of:

storing in a database of the proxy the configurable configuration.

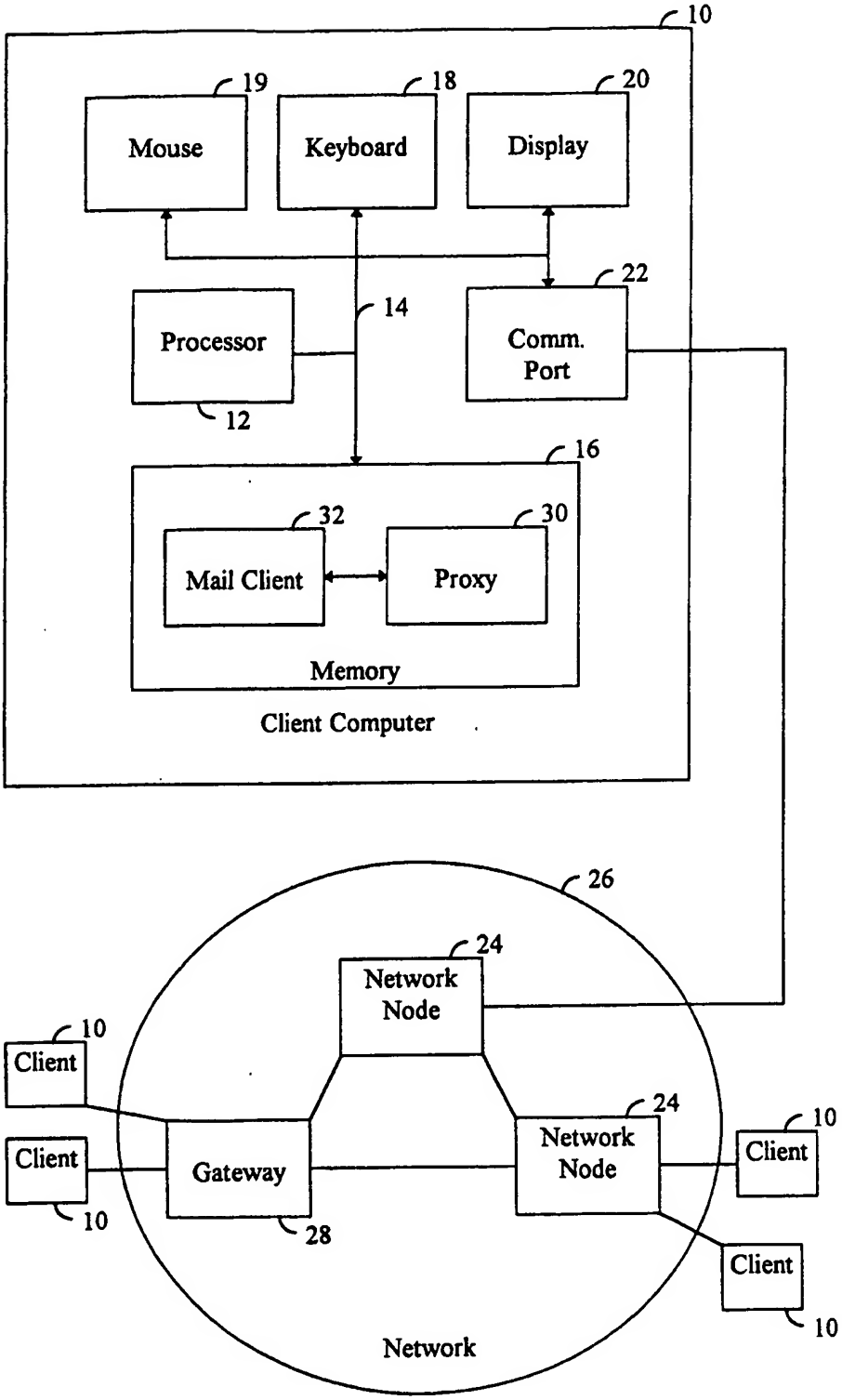

10
19
Mouse
18
Keyboard
20
Display
22
14
Processor
12
Comm. Port
16
32
Mail Client
30
Proxy
Memory
Client Computer
26
24
Network Node
10
Client
Client
10
Gateway
28
24
Network Node
10
Client
Client
10
Network

FIG. 1

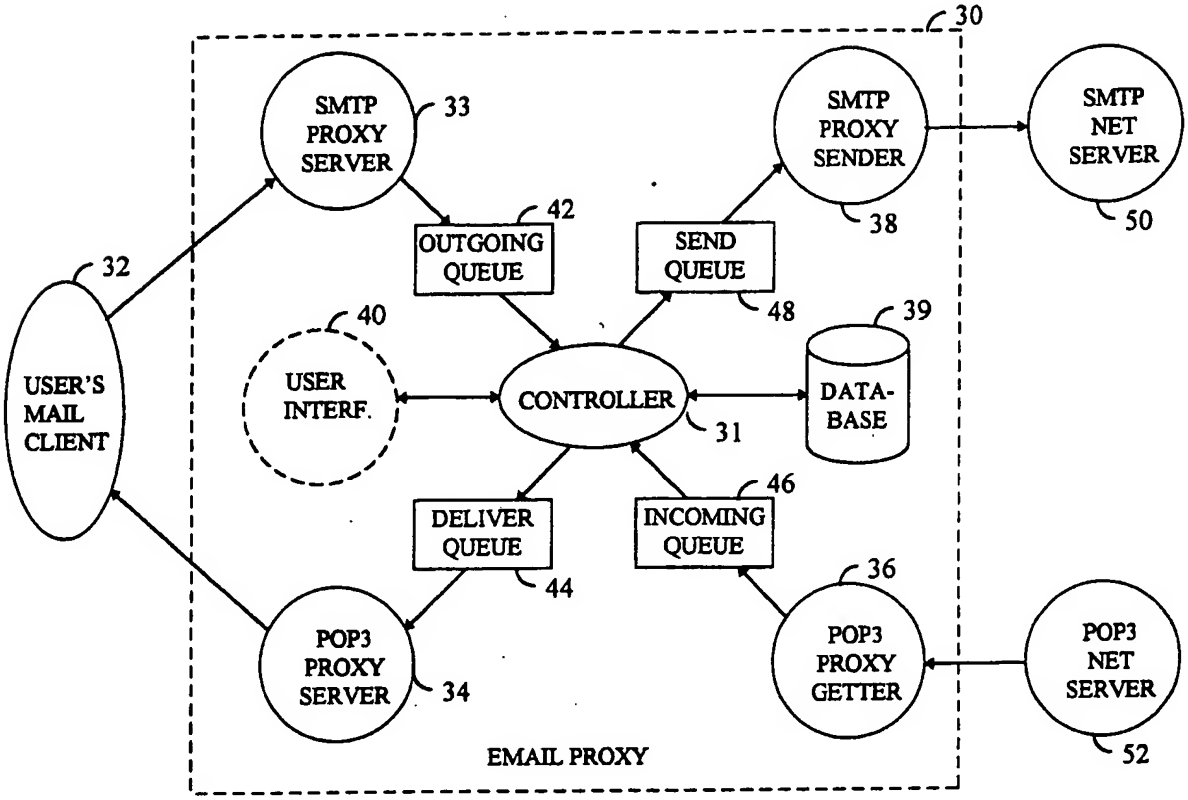

30
SMTP PROXY SERVER
33
SMTP PROXY SENDER
SMTP NET SERVER
38
50
42
OUTGOING QUEUE
SEND QUEUE
48
32
40
39
USER'S MAIL CLIENT
USER INTERF.
CONTROLLER
31
DATA-BASE
46
DELIVER QUEUE
INCOMING QUEUE
44
36
POP3 PROXY SERVER
34
POP3 PROXY GETTER
POP3 NET SERVER
52
EMAIL PROXY

FIG. 2

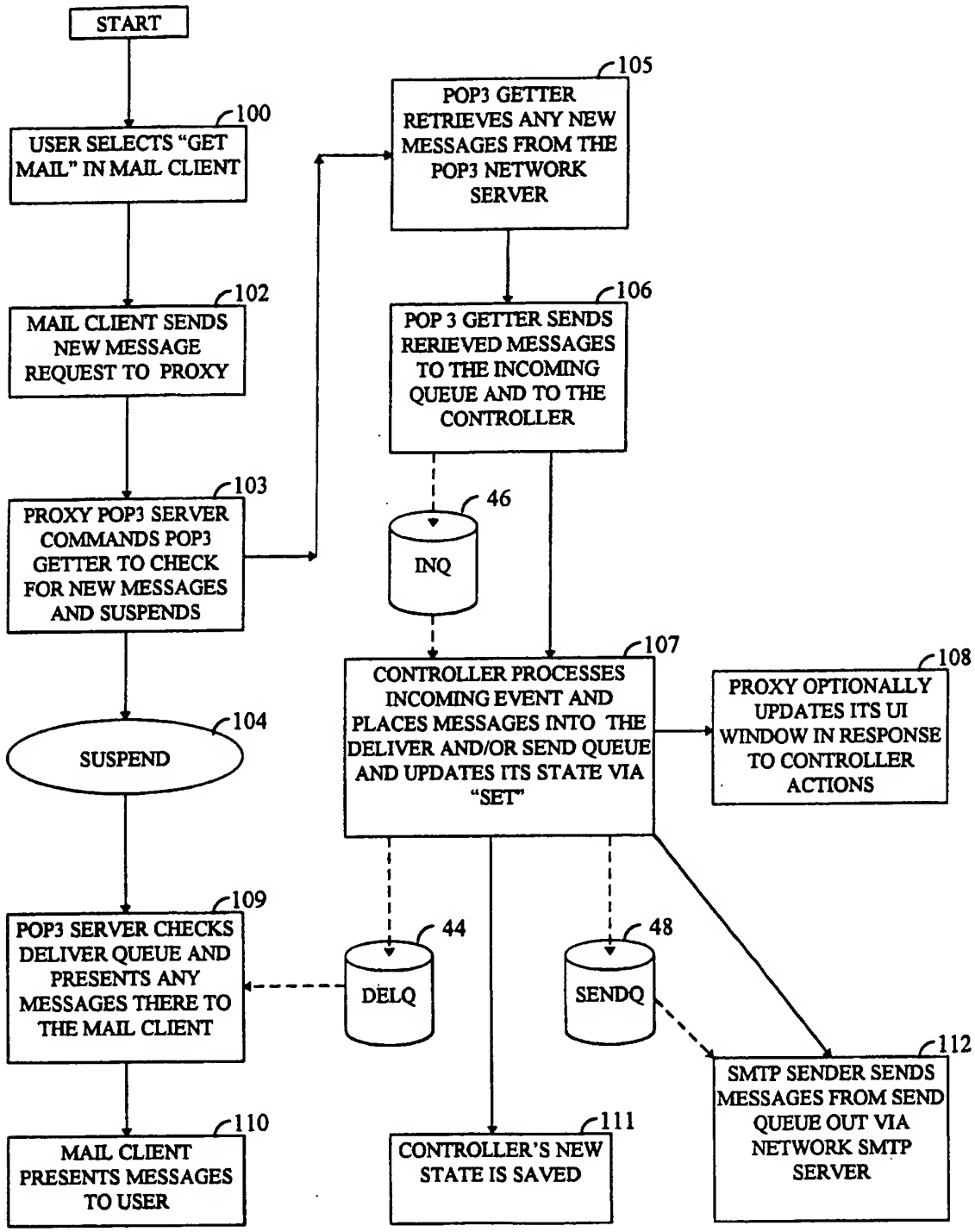

START
100
USER SELECTS "GET MAIL" IN MAIL CLIENT
102
MAIL CLIENT SENDS NEW MESSAGE REQUEST TO PROXY
103
PROXY POP3 SERVER COMMANDS POP3 GETTER TO CHECK FOR NEW MESSAGES AND SUSPENDS
104
SUSPEND
109
POP3 SERVER CHECKS DELIVER QUEUE AND PRESENTS ANY MESSAGES THERE TO THE MAIL CLIENT
110
MAIL CLIENT PRESENTS MESSAGES TO USER
105
POP3 GETTER RETRIEVES ANY NEW MESSAGES FROM THE POP3 NETWORK SERVER
106
POP 3 GETTER SENDS RERIEVED MESSAGES TO THE INCOMING QUEUE AND TO THE CONTROLLER
46
INQ
107
CONTROLLER PROCESSES INCOMING EVENT AND PLACES MESSAGES INTO THE DELIVER AND/OR SEND QUEUE AND UPDATES ITS STATE VIA "SET"
108
PROXY OPTIONALLY UPDATES ITS UI WINDOW IN RESPONSE TO CONTROLLER ACTIONS
44
DELQ
48
SENDQ
111
CONTROLLER'S NEW STATE IS SAVED
112
SMTP SENDER SENDS MESSAGES FROM SEND QUEUE OUT VIA NETWORK SMTP SERVER

FIG. 3

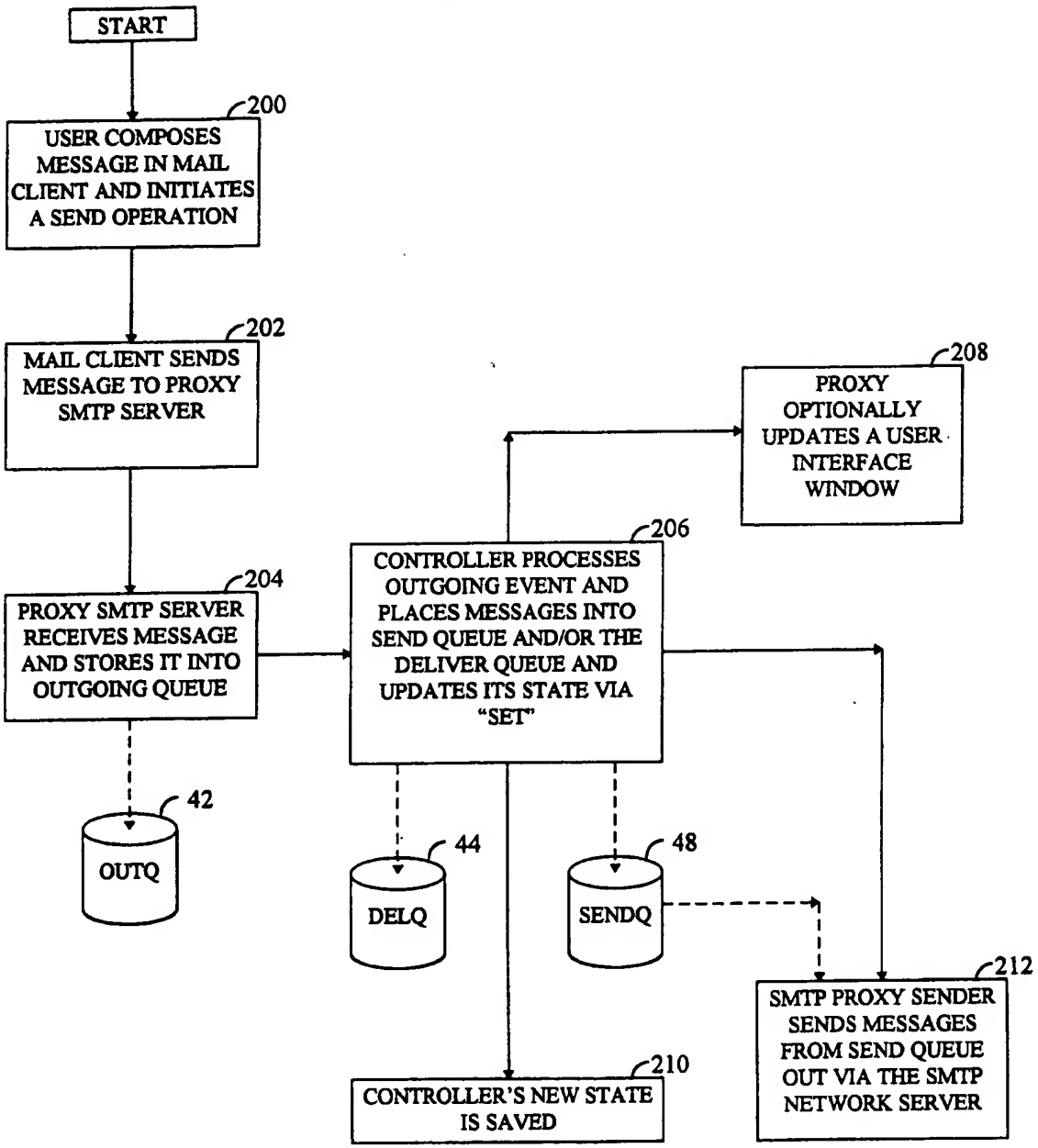

START
200
USER COMPOSES MESSAGE IN MAIL CLIENT AND INITIATES A SEND OPERATION
202
MAIL CLIENT SENDS MESSAGE TO PROXY SMTP SERVER
204
PROXY SMTP SERVER RECEIVES MESSAGE AND STORES IT INTO OUTGOING QUEUE
42
OUTQ
206
CONTROLLER PROCESSES OUTGOING EVENT AND PLACES MESSAGES INTO SEND QUEUE AND/OR THE DELIVER QUEUE AND UPDATES ITS STATE VIA "SET"
208
PROXY OPTIONALLY UPDATES A USER INTERFACE WINDOW
44
DELQ
48
SENDQ
212
SMTP PROXY SENDER SENDS MESSAGES FROM SEND QUEUE OUT VIA THE SMTP NETWORK SERVER
210
CONTROLLER'S NEW STATE IS SAVED

FIG. 4

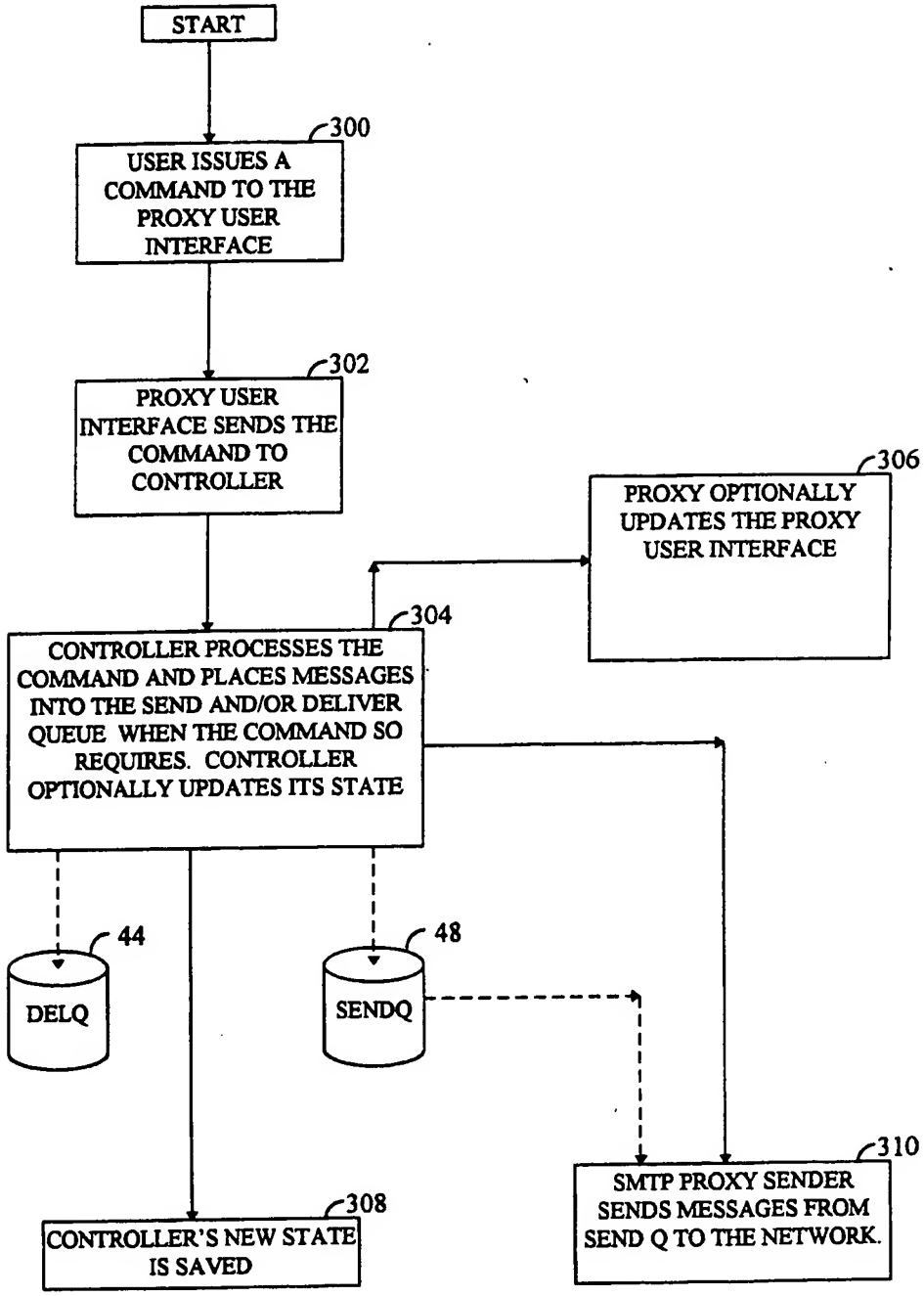

START
300
USER ISSUES A COMMAND TO THE PROXY USER INTERFACE
302
PROXY USER INTERFACE SENDS THE COMMAND TO CONTROLLER
306
PROXY OPTIONALLY UPDATES THE PROXY USER INTERFACE
304
CONTROLLER PROCESSES THE COMMAND AND PLACES MESSAGES INTO THE SEND AND/OR DELIVER QUEUE WHEN THE COMMAND SO REQUIRES. CONTROLLER OPTIONALLY UPDATES ITS STATE
44
DELQ
48
SENDQ
310
SMTP PROXY SENDER SENDS MESSAGES FROM SEND Q TO THE NETWORK.
308
CONTROLLER'S NEW STATE IS SAVED

FIG. 5